Válasz Dr. Szolgay Péter professzor úr bírálatára

MTA értekezés címe: Modelltranszformációk validálása és alkalmazása

Megköszönöm Dr. Szolgay Péter professzor úrnak az értékes észrevételeit, a további részletekre rákérdező kérdéseit, és természetesen az eredményeimet illető elismerő szavait. Külön köszönöm a modelltranszformációs eljárások osztályozásához és a harmadik téziscsoporthoz kapcsolódó elismerő szavait, valamint a tézisek megfogalmazásával kapcsolatban tett javaslatait. Köszönöm a felmerült kérdéseket és megjegyzéseket, melyek megválaszolásával lehetőséget kaptam az eredmények néhány további részletének kifejtésére és esetenként további részeredmények említésére is.

A dokumentumban dőlt betűtípussal szedve szerepelnek a bírálatból átvett kérdések, majd ezt közvetlenül követik a válaszaim.

---

1. kérdés: *A szoftver modellezés és a szakterület specifikus modellek magas szintű specifikációból indulnak és a cél a kód generálás. A disszertáció ismerteti a különböző modell transzformációs megoldásokat és ezek közül a gráf újra írásos transzformációt ismerteti részletesebben. Hiányolom ennek a transzformációnak a pontos definícióját.*

Hiányolom a gráf pontos definícióját, mik a szögpontok és mik az élek?

A címkézett irányított gráfok (labeled directed graphs) és gráftranszformáció definícióját a következő mű alapján használom, a definíciókat angol nyelven adom meg.

> G. Rozenberg (ed.), Handbook on graph grammars and computing by graph transformation: Foundations 1, World Scientific, Singapore, 1997.

A címkézett irányított gráf definíciója:

Definition (*labeled directed graphs, LDG*). Let Ω_V and Ω_E be two given alphabet for node and edge labels, respectively. Then the LDG is a six-tuple: $G = \langle G_V, G_E, s^G, t^G, lv^G. le^G \rangle$, where G_V is the set of vertices, G_E is the set of edges; s^G and t^G are the source and target functions (s^G, t^G : $G_E \rightarrow G_V$), which map an edge to its source and target vertex, respectively; and finally, lv^G : $G_V \rightarrow \Omega_V$ and le^G : $G_E \rightarrow \Omega_E$, which assign a label to a vertex and an edge from the appropriate alphabet.

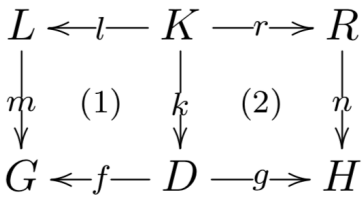
A gráfújraírási szabály definíciója:

Definition (*graph production rule*). A graph production rule $p : L \leftarrow_l K \rightarrow_r R$ is composed of a production name p, and a pair of injective graph morphism: $l : K \rightarrow L$, and $r : K \rightarrow R$. The graphs L, K and R are called the left-hand side, the interface graph, and the right-hand side graph of p, respectively.

A gráfújaraírási szabályokból felépülő gráftranszformáció definíciója:

Definition (*direct graph transformation*). Given a graph production $p : L \leftarrow_l K \rightarrow_r R$ and a graph G with a graph morphism $m : L \rightarrow G$, called a match. A **direct graph transformation**

$G \Rightarrow_{p,m} H$ from G to a graph H is given by the following double pushout (DPO) diagram, where (1) and (2) are pushouts. We say p is applicable to G via m if pushouts (1) and (2) exist.



Az ábrához kapcsolódó kiegészítés:

The DPO approach accomplishes rule firing in two steps: after finding a redex (the part of the host graph matched by the rewriting rule), the first step removes the elements (vertices and edges) from the redex which are in the redex, but not in RHS graph. This modified redex is referred to as interface graph. Then as a second step the elements of RHS graph not in the interface graph but in RHS graph are glued to the interface graph.

The illustration of direct derivation: *L, K, R* are the left-hand side, the interface and the right-hand side graph; *G* and *H* are the graphs before and after the rule firing and *D* corresponds to *K; l, r, f, g, m, k, n* are inclusions.

A kategóriaelméleten alapuló algebrai gráfújraírási megközelítéseknek két ága van: a fenn bemutatott double pushout (DPO), valamint a single pushout (SPO) megközelítés. Az SPO megközelítés egy lépésben végzi el az újraírás műveletet. A DPO megközelítés esetén egy interfész gráf keletkezik a nem szükséges élek és csomópontok törlését követően, majd külön lépésben történik az új csomópontok és élek létrehozása. A DPO ágazat bizonyult mind a kutatók ráépülő további munkáit, mind a gyakorlati irányba történő alkalmazhatóságot tekintve meghatározónak. Ennek megfelelően én is a DPO ágazat elméleti eredményeire alapozottan végzem a kutatómunkámat.

---

2. kérdés: *Definiálja a verifikációs/validációs eljárások komplexitásának a fogalmát.*

A verifikációs/validációs eljárások komplexitása azt a lépésszámot jelenti, amely az adott eljárást használva minimálisan szükséges a kiértékelés elvégzéséhez.

---

3. kérdés: *Milyen a 3.5 fejezet algoritmusainak (1.3 tézis) időfüggése?*

A 3.5 fejezetben két algoritmust mutattam be a deklaratív szabályokkal és vezérlésmodellel definiált modelltranszformációk tesztvezérelt validációjára. A módszer mindkét algoritmusa lehetővé teszi, hogy a tesztelendő modelltranszformációt teljesen vagy részben lefedő bemeneti modelleket, illetve modellhalmazokat állítsunk elő a bemeneti metamodell és a modelltranszformáció definíciója (azaz a transzformációs szabályok definíciói, valamint a vezérlésfolyam modell) alapján.

A kutatómunka során kidolgozott algoritmusok esetén a feldolgozáshoz szükséges minimális lépésszámot vizsgáltam.

Az érvényes bemeneti modellek generálásához szükséges GENERATIONINPUT-BASIC algoritmus számítási komplexitása $O(\sum_{i=1}^{k} v_i^2 + v_i * v_m)$ ahol k a transzformációs szabályok száma, v_i a csomópontok száma az i-edik transzformációs szabályban, valamint v_m a csomópontok száma a generált modell metamodelljében (a transzformáció bemeneti modelljének a metamodelljében). Az összefüggés által leírt összeg első része a mintaillesztés költsége, a második rész pedig a modellrészek összefűzésének költsége.

Az érvényes bemeneti modellek generálásához szükséges GENERATIONINPUT-ADVANCED algoritmus számítási komplexitása $O\left(\sum_{i=1}^{k} 2 * v_i^2 + v_i * v_m + k * v_i^2\right) = O\left(\sum_{i=1}^{k} (2+k) * v_i^2 + v_i * v_m\right)$, ahol k a transzformációs szabályok száma, v_i a csomópontok száma az i-edik transzformációs szabályban, valamint v_m a csomópontok száma a generált modell metamodelljében (a transzformáció bemeneti modelljének a metamodelljében). Az összefüggés által leírt összeg első része a mintaillesztés költsége, a második rész a modellrészek összefűzésének költsége, a harmadik rész pedig a szabály jobb oldali (RHS) lehetséges variációinak kalkulálási költsége (az egyenlőség jobb oldalán a képlet egyszerűsítése történt meg).

Az algoritmusok elemzésében a következő cikk tartalmaz további részleteket:

> L. Lengyel, H. Charaf, Test-driven verification/validation of model transformations, Frontiers of Information Technology & Electronic Engineering 16:(2), pp. 85-97, 2015.

---

4. kérdés: *A mobil rendszerek energia-hatékony kezelésére ad javaslatot. Magasszintű közelítéssel él és a kommunikációt optimalizálja. Kérdés, hogy van-e lehetőség a rendszer egyéb folyamatainak optimalizálására is az energiafelhasználást illetően?*

Igen, a rendszer más folyamatai is optimalizálhatók az energiafelhasználást illetően. Ilyen folyamatok például a mobil eszközök különböző hardver elemeinek (kamera, kijelző, vagy biometrikus azonosító eszközök) a kezelése. Fő optimalizálási lehetőség a mobil eszközökön futó algoritmusok futtatási költségének mérséklése, illetve kiváltása azzal, hogy a feldolgozási műveleteket a mobil eszköz helyett a felhőben végezzük el. Itt mérlegelendő, hogy mely feldolgozások esetén jelent maga a feldolgozás, illetve a művelet kihelyezéséhez szükséges kommunikáció több energiát. Ezen a területen is végeztünk kutatómunkát kollégáimmal. Egy ehhez a tematikához tartozó publikáció:

> K. Fekete, T. Kárai, I. Albert, H. Charaf, L. Lengyel, Mobiltelefonok energiafogyasztásának vizsgálata Windows Phone platformon, Elektrotechnika 106:(3) pp. 12-24. (2013)

Az értekezésben bemutatott eredmény ugyanakkor csak a mobil eszközök kommunikációjának optimalizálását részletezi. Ezen a területen volt lehetőségem több hónapon keresztül együtt dolgozni az Aalto University két kutatójával. Jukka K. Nurminen és Matti Siekkinen a mobil kommunikációs területet kutatásaik révén alaposan megismerték, széleskörű méréseikkel meghatározó eredményeket értek el. Lehetőségem volt az értekezésben bemutatott módon a mobil eszközök energiahatékony működését meghatározó tulajdonságok szoftvermodellek szintjén történő definiálására, valamint modellvezérelt feldolgozására. A módszer alkalmazása révén a szoftverrendszerek energiahatékonysági aspektusa már a modellek szintjén is

megjelenik, ezáltal teljesebb képet formál az alkalmazás elvárt működéséről, támogatja a pontos analízist, valamint hozzájárul a verifikációs és validációs módszerek teljességéhez.

A területhez kapcsolódó további részleteket foglalok össze a 6-os kérdés megválaszolásánál.

---

5. kérdés: *A 4-1 ábra alapján működő modell alapú szoftverfejlesztés tesztelése költséghatékony megoldást eredményez. Mit lehet mondani a hatékonyságjavulás mértékéről?*

A jelen kérdés megválaszolása során ezen a helyen a módszer teszteléshez kapcsolódó lépéseit és annak hatékonyságnövelő hatásait foglalom össze. A 7-es számú kérdés megválaszolása során további részleteket foglalok össze a teljes módszerről és a kapcsolódó hatékonyságnövekedésről.

Automatikus teszteset generálás: Az összetett szoftverrendszerek összköltségének jelentős részét a tesztelésre és hibajavításra fordítjuk. Az egyre komplexebbé és integráltabbá váló szoftverek szükségessé teszik, hogy a tesztelést minél nagyobb mértékű automatizmussal támogassuk, valamint, hogy a fejlesztéseknek minél korábbi szakaszaiban derítsük fel a hibákat. Az automatikus tesztelés megvalósításával a modell, vagy kód változásakor nem szükséges a teszteseteket manuálisan karbantartani, nincs külön teszttervezési költség, továbbá jól mérhető a tesztek által lefedett programrészek nagysága is. *(a hatékonyságjavulás 1. komponense)*

Általánosan igaz, hogy csak programkódból nehezen állíthatók elő a tesztesetek, azonban, ha a kódot modellből generáljuk vagy modell definiálja az elvárt funkcionalitást, akkor lehetőség van modellalapú tesztelési megközelítéseket használni. Ilyenkor a tesztesetek közvetlenül a modellből származnak. A szakterületi modell által leírt logika tesztelésével már a tervezési fázisban felismerhetjük a hibák egy részét. *(a hatékonyságjavulás 2. komponense)*

A valóban hasznos (releváns) tesztforgatókönyvek generálását biztosító kutatómunka célja azon automatizmusok kidolgozása volt, amelyek a követelményspecifikációt leíró modellek által meghatározott rendszer funkcionalitását teljes mértékben lefedő tesztforgatókönyveket generálnak. Jelentős hatékonyságnövekedést jelent, hogy az esetlegesen igen nagy számban megjelenő irreleváns, valós környezetben elő nem forduló esetek, nem foglalnak tesztelési kapacitást. A tesztesetek állapotterének egyszerűsítése lehetővé teszi az összes releváns forgatókönyv ellenőrzését idő és költséghatékony módon. *(a hatékonyságjavulás 3. komponense)*

A kutatómunka eredménye egy olyan eljárás, amelynek segítségével a felhasználói forgatókönyvek alapján a forgatókönyvekhez tartozó valós teszteset leírások automatikusan előállíthatók. Tapasztalataim szerint a fenti három faktor együttesen akár a harmadára /negyedére csökkentheti a tesztelési feladatok elvégzéséhez szükséges időt és erőforrásokat.

---

6. kérdés: *Mekkora energia hatékonyság javulás várható a 4.5 fejezet eredményének köszönhetően?*

A 4.5 fejezetben bemutatott eredményemnek a legfontosabb aspektusa, hogy az alkalmazások energiahatékonyságát befolyásoló attribútumokat is a modellek absztrakciós szintjére tudjuk

emelni, ezzel bővíteni lehet a szoftverrendszer pontosabb elemzési, valamint verifikálási és validálási lehetőségeit.

A fejezetben tárgyalt, az általam kidolgozott modellvezérelt megoldás révén kontrollált hatékonyabb energiafelhasználást lehetővé tevő módszerek (Delayed Transfer, Bursty Transfer, Compressed Transfer), a felhasználási céloktól és elvárásoktól függően, saját méréseink alapján 10-45%-os energiamegtakarítást tudnak elérni az adattovábbítás területén.

---

*7. **kérdés:** Az elért kutatási eredmények mennyiben (esetleg paraméterekkel alátámasztva) javították a szoftver-fejlesztés hatékonyságát?*

A szakterület-specifikus modellezés és modellfeldolgozás előnyeit felhasználva a szoftverprojektek minőségét biztosító és hatékonyságát növelő modellalapú fejlesztési módszer a fejlesztés teljes folyamatát támogatja: felmérés, követelményelemzés, fejlesztés, tesztelés és dokumentálás (**Error! Reference source not found.**. ábra a tézisfüzetben, illetve 4-1. ábra az értekezésben). A követelményelemzési folyamat eredménye a követelményspecifikáció, ami modellek egy halmaza, melyek jól érthetők az adott szakterület ismerői számára, ugyanakkor kellően formálisak az automatikus feldolgozáshoz. A követelményspecifikáció modelljeiből a modellfeldolgozók generálják a szoftvertermékek (forráskód, konfigurációs fájlok, további termékek) megfelelő részeit, a tesztelési forgatókönyveket és a dokumentációt. A jóváhagyott követelményspecifikáció képezi minden további folyamat és szoftvertermék alapját. A forráskód és a tesztforgatókönyvek, valamint a verifikálás és a validálás is a követelményspecifikációban rögzített elvárásokból indul ki. A módszer egy iteratív fejlesztési folyamatot eredményez: a tapasztalatok és a visszajelzések visszavezetésre kerülnek a követelménymodellekbe, majd a generálás révén a következő iterációban jutnak érvényre.

A megoldás fő hozzáadott értéke a hatékonyságot és a magas minőségű szoftvertermék előállítását támogató folyamat létrejötte. Mindezt úgy értük el, hogy évtizedes tapasztalatra alapozottan készültek el azok a forráskód szintű osztálykönyvtárak, amelyek függvényeit a generált kód használja. Tehát a funkciók jelentős része általános formában megtalálható a támogató osztálykönyvtárakban és keretrendszerekben. Ezen komponensek minőségét kvalifikált fejlesztők részvétele, valamint a kódok több projektben történő felhasználása, ebből következően alapos tesztelése fémjelzi. A forráskód generálás során jól paraméterezett függvényhívások sorozata kerül generálásra. Tehát a generálás az osztálykönyvtárak feletti, az adott követelményekhez igazodó specifikus réteg. A teljes megvalósítást tekintve ez a 30-40%-át jelenti a termékbe/szolgáltatásba kerülő forráskódnak.

A hozzáadott értékben a hatékonyságot a számos alkalommal újra használható modellfeldolgozók jelentik. Ezek a modellfeldolgozók illeszkednek a szakterületi nyelvekhez és ismerik az osztálykönyvtárakat, melyek funkcionalitásán alapszik az általuk generált projekt-specifikus kódréteg.

A szoftverfejlesztési projektek elemzése két módon történt meg:

- Adott teljesítmény (szoftver funkciók elkészítése) kisebb létszámú csapattal, ahol 40-50%-os méretű csapat tudta a módszer alkalmazásával az előre meghatározott fejlesztési ütemtervet tartani.

- Megegyező méretű csapatok esetén, pedig hasonlóan, valamivel fele idejű átfutás alatt (45-50% időkeretben) történt meg az adott funkcionalitás elkészítése és a minőségi kritériumokat teljesítő tesztelési szakasz elvégzése.

Minél hosszabb a szoftvertermék életciklusa (a karbantartást és továbbfejlesztést is figyelembe véve), annál jelentősebb a kidolgozott eredmények hatása a hatékonyságra. A tapasztalatok azt mutatják, hogy adott méretnél nagyobb rendszerek fejlesztése esetén érdemes alkalmazni (pl. legalább 6 db 2 hetes iteráció), amelyek gondos modellezést, pontos munkafolyamatokat, továbbá részletes dokumentációt és mélytesztelést igényelnek. Számos ilyen célterület van, például a banki alkalmazások, közigazgatás, biztosító társaságok, gyógyszergyárak rendszerei, vállalatirányítási rendszerek, stb.

Általánosságban elmondható, hogy mind az ügyfél, mind a szolgáltató/fejlesztő érdekelt a rendszerkövetelmények pontos meghatározásában. Projektek igazolták, hogy az előzetes fázisban végzett körültekintő erőfeszítés, a modellezés és a tervezés, jelentősen csökkentik a kockázatokat (idő, költségek, nem használt funkciók száma). Ez volt a fő cél a módszer kidolgozása és alkalmazása során.

---

8. kérdés: *A 3.2 tézis eljárást adott a szakterület-specifikus tulajdonságok ellenőrzésére a modellfeldolgozás során. Bebizonyította (kísérleti v. elméleti úton?), hogy a validáló szabályok nem módosítják a kimeneti modellt.*

A 3.3 tézis algoritmusokat adott a validáló kényszerek modularizált kezelésére. Az eljárás kísérleti vagy formálisan is bizonyítható?

A 3.2 tézis állításainak bizonyítása formális módon történt. A kapcsolódó algoritmusokkal és állításokkal kapcsolatos részleteket a következő publikációm tartalmazza:

> L. Lengyel, The Role of Graph Transformations in Validating Domain-Specific Properties, International Journal of Computer Engineering and Technology 3:(3), pp. 406-425, 2012.

A bizonyításokat a válasz mellékleteként is csatolom.

A bizonyítások során a kidolgozott algoritmusok lépéseit követem. Mind a 8 állítás bizonyítása során, lépésről-lépésre megvizsgálom az elvégzett műveletet, a műveletvégzés körülményeit (paramétereket, az újraírási szabályok bal és jobb oldali struktúráját, az újraírási szabályokhoz rendelt kényszer feldolgozó, illetve kiértékelő jellegét), elemzem a művelet eredményét és ez alapján vonom le a következtetéseket, illetve hozom meg a döntéseket.

A 3.3 tézisben bemutatott eljárásom kísérleti. A megoldás tesztelése és igazolása során egyrészt a validáló kényszerek modularizált kezelését támogató algoritmusok nélkül kerültek futtatásra a modelltranszformációk, másrészt az algoritmusok (az ismétlődő és átszövő kényszereket azonosító színező algoritmus, valamint a színezés alapján a transzformációkból az ismétlődő és átszövő kényszerek kinyerését és a visszaszövésükhöz szükséges referencialisták előállítását támogató algoritmusok) alkalmazását követően. A módszert és algoritmusait alkalmazva, a céloknak megfelelően, a modularizált kezelés biztosítja a kényszerek hatékony és konzisztens kezelését, továbbá a kidolgozott algoritmusaimmal támogatott módszer segítségével a modellfeldolgozás eredménye megegyezik az algoritmusok nélküli futtatás eredményével.

A bírálat külön szakaszban foglalkozik a tézisek értékelésével. A felmerült észrevételeket köszönöm. A következő bekezdésekben igyekszem azokra megnyugtató válaszokat adni.

„A pályázó 1.2 Tézise a szabály alapú rendszerek validálására ad módszert. Ez a tézis megítélésem szerint Charaf Hassan 1.4 tézisével azonos állítást közöl, amelyet egy közös, kétszerzős cikkben publikáltak. Ezt a tézist így nem tudom elfogadni új tudományos eredményként.”

Az 1.2 tézis keretében célom egy általános érvényű szabály alapú validálási módszer definiálása volt. Ez a megközelítés eltér Charaf Hassan egy szűk területre irányuló állításától. Az indoklásomat a következő formában bontom ki:

Charaf Hassan 1.4 tézisének címe: „Többplatformos modelltranszformációk futásidejű validálása”. Charaf Hassan tézise azt a módszert fogalmazza meg, mint eredményt, amely támogatja a gráfújraírás-alapú modellfeldolgozás validálását egy speciális esetben, nevezetesen a „többplatformos fejlesztési környezetekben”.

Az én 1.2-es tézisem két dolgot fogalmaz meg:

- Általában a szabályalapú rendszerekhez kapcsolódóan a futásidejű validálás: szabályalapú rendszereket gráfújraírási rendszerek igényének megfeleltetve kidolgoztam a szabályalapú rendszerek validálásának módszerét.
- Kidolgoztam továbbá a verifikációs/validációs megoldások komplexitásának egyszerűsítő módszerét.

Charaf Hassan 1.4-es tézise a fentiek szerint egy szűk területre, egy speciális esetre vonatkozik. A többplatformos modelltranszformációk esetén generált modell minden esetben, minden platform esetén forráskódot reprezentáló fa struktúra.

Az én tézisemben a szabályalapú rendszerek esetén a modellfeldolgozók tetszőleges forrás és cél modellek közti leírást jelentenek. Ennek megfelelően a nagyobb problématér miatt a gyakran összetettebb modellfeldolgozásokat is számba kell venni. Emiatt jelenik meg a tézisben a verifikációs/validációs megoldások komplexitásának egyszerűsítő módszere is.

Az említett cikk első szerzője én vagyok.

A 2.2, 2.3, 2.4 és 3.1 tézisekhez kapcsolódóan: a tézis megfogalmazása nem felel meg az elvártaknak (egyes szám első személyben kell kimondani).

A tudományos eredményeimet három téziscsoportba strukturáltan 10 tézispontban foglaltam össze. A tézispontok többségét egyes szám első személyben fogalmaztam meg. A fenti esetekben többes szám első személyt vagy passzív szerkezetet használtam, melynek oka:

A 2.2 és 2.3 pontok esetén a szerényebb jellegű hozzáállás, mert ezen pontokban több implementálási feladat elvégzésében, segítséget kaptam PhD hallgatóimtól. A tudományos eredményeket illetően a sajátomnak tartom mindkét tézist.

A 2.4 tézisnél, az „energiahatékony működést meghatározó tulajdonságok modellvezérelt kezelésének módszere” területen, ahogy a 4-es számú kérdésre adatt válaszokban leírtam, külföldi kutató kollégák adták a mobilkommunikációs hátteret a megoldáshoz. A tézisem a modellvezérelt megoldásra vonatkozik, amely a saját munkám eredménye. A mobilkommunikációt mint szakterületet használva, annak specialitásait figyelembe véve (itt volt meghatározó szerepük a külföldi kutató kollégáknak) dolgoztam ki a bemutatott eredményt.

A 3.1 tézis kimondásánál a tézis első felében egyes szám első személyt használok:

„Elméleti és gyakorlati alapjait dolgoztam ki a szakterület-specifikus tervezési minták támogatásának metamodellező környezetben a következőkkel jellemezve: Megmutattam, hogy a modelltranszformációs szabályokat metamodell szintjén definiálva példányosítás relációval illeszthető a transzformációs szabály bal oldala, valamint a jobb oldalnak példánya lesz a kimeneti modell adott része.”

A tézis második felénél figyelmetlenség miatt használtam a passzív szerkezetet („Bizonyításra került, hogy ...” és „Igazolásra került, hogy ...”). A tézis pontosított változata a következő:

Elméleti és gyakorlati alapjait dolgoztam ki a szakterület-specifikus tervezési minták támogatásának metamodellező környezetben a következőkkel jellemezve: Megmutattam, hogy a modelltranszformációs szabályokat metamodell szintjén definiálva példányosítás relációval illeszthető a transzformációs szabály bal oldala, valamint a jobb oldalnak példánya lesz a kimeneti modell adott része. Bebizonyítottam, hogy metamodelljének megfelelően a tranzitív tartalmazást teljesítő modell nem feltétlenül részleges példánymodell. Igazoltam, hogy a metamodell részleges példánymodelljei a példányosítás relációt csak (i) a relaxált multiplicitás és kardinalitás, (ii) a relaxált lógó élek, valamint (iii) a nemteljes attribútumok esetén sértik meg.

Még egyszer megköszönöm a Bíráló munkáját, kutatómunkámat elősegítő hasznos kérdéseit és észrevételeit, valamint a bírálat elkészítésére szánt energiáját, idejét. Megtisztelő számomra, hogy több ponthoz kapcsolódóan dicsérte annak bemutatást és hasznosságát, valamint kiemelte alkalmazásuk jelentőségét. Bízom benne, hogy a fentiekben megnyugtató válaszokat tudtam adni a felmerült kérdésekre.

Budapest, 2019. június 10.

Lengyel László

Melléklet: Bizonyítások a 3.2-es tézishez

Proposition 1. A validation transformation rule generated with algorithm GENERATEVALIDATIONTRANSFORMATIONRULE does not modify the output model.

Proof. The role of a validation transformation rule is the validation of the required conditions. Based on the algorithm GENERATEVALIDATIONTRANSFORMATIONRULE, that generates the validation transformation rules, we know that the LHS and the RHS of the validation rules have the same pattern. No additional imperative code is added to the validation rule. Therefore, the generated rule does not modify the output model.

The only difference between the LHS and RHS of the validation rule is the constraint propagated only to the LHS of the rule. The role of the constraint is to refine the matching, i.e. after finding the structure, additional conditions are validated. Therefore, the generated validation rule does not have any effect on the output model.

Proposition 2. Validation transformation rules generated with algorithm GENERATEVALIDATIONTRANSFORMATIONRULE are not necessarily valid partial instances of the output metamodel.

Proof. Generated validation transformation rules validate output model-related domain-specific properties, therefore they always instantiate the output metamodel.

Based on the definition of *Partial instantiation* (Definition 3), a valid partial instance, in the current case the generated rule (both LHS and RHS), can be extended to a valid instance of the metamodel. During the pattern extension both modification and deletion are forbidden; only setting uninitialized attribute values and adding new elements are permitted. In Figure 7a, an example invalid partial instance is presented. It contains the pattern both the LHS and RHS patterns of a generated validation rule - proving that there exists a case when a rule is not a partial instance of the metamodel.

4.2 Extending model transformations with validation transformation rules

We use the algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES to extend the control flow model of the transformation with the generated validation transformation rules. The input parameters of the algorithm define the transformation *T* that should be extended, a condition list containing SCs and NSCs, and the list of the validation points where SCs and NSCs should be validated. The algorithm creates a flow final node (line 2) then, using a loop, processes the validation points (line 3-25). For each validation point the SC or NSC condition is selected from the *conditionList* (line 4). Next, based on the representation type of the condition (*ConstraintBased*, *PatternBased*, or *Hybrid*), the algorithm creates a validation transformation rule. In the case of *ConstraintBased* representation type, using the constraint of the condition, the validation rule is created with algorithm GENERATEVALIDATIONTRANSFORMATIONRULE (line 6-7). Otherwise, for conditions with representation type *PatternBased* and *Hybrid* the validation rule is created based on the pattern of the condition (line 10). Both the LHS and the RHS of the new validation rule is initialized with the pattern of the condition. For *Hybrid* type conditions, the constraint of the SC or NSC is propagated to the LHS of the validation rule (line 12-14). Then the algorithm assigns the new rule to the transformation *T* (line 15), and arranges the necessary flow edge modifications (line 16-24). During the creation and rearrangement of the flow edges we differentiate the success type of the conditions, i.e. SCs and NSCs are handled in a different way. Figure 12 illustrates this part of the algorithm. In both of the cases the new validation rule is inserted after the transformation rule selected by the *validationPoint*. The difference is in the type of the flow edges beginning from the validation rule. In the case of SCs, a *Fail* type edge goes to the *flowFinal* and *Success* type edges points to the original target rules or to the end rule. Contrarily, in the case of NSCs, a *Success* type edge goes to the *flowFinal* and *Fail* type edges point to the original target rules or to the end rule. Finally, the extended transformation definition is returned.

Algorithm 2. Pseudo code of the EXTENDTRANSFORMATIONWITHVALIDATIONRULES algorithm

```
1: EXTENDTRANSFORMATIONWITHVALIDATIONRULES (Transformation T, List conditionList,
   List validationPointList) : Transformation
2: FlowFinal flowFinal = T.CreateFlowFinal()
3: for all ValidationPoint validationPoint in validationPointList do
4:   Condition condition = conditionList.Get(validationPoint.ConditionReference)
5:   TransformationRule validationRule = NULL
6:   if (condition.RepresentationType == RepresentationType :: ConstraintBased) then
7:     validationRule = GENERATEVALIDATIONTRANSFORMATIONRULE(condition.Constraint)
8:   else
9:     // executed for PatternBased and Hybrid type conditions
10:    validationRule = CREATETRANSFORMATIONRULE(LHS=condition.Pattern, RHS=condition.Pattern)
```

11: **end if**
12: **if** (*condition:RepresentationType == RepresentationType :: Hybrid*) **then**
13: PROPAGATECONSTRAINT(*validationRule.LHS, condition.Constraint*)
14: **end if**
15: *T.AddTransformationRule(validationRule)*
16: **if** (*validationPoint:SuccessT ype == SuccessType :: SC*) **then**
17: CREATEFLOW(*validationRule, flowFinal, FlowType :: Fail*)
18: MODIFYFLOWS(*validationPoint.TransformationPoint, validationRule, FlowType :: Success*)
19: **else**
20: // executed for NSCs
21: CREATEFLOW(*validationRule, flowFinal, FlowType :: Success*)
22: MODIFYFLOWS(*validationPoint.TransformationPoint, validationRule, FlowType :: Fail*)
23: **end if**
24: CREATEFLOW(*validationPoint.TransformationPoint, validationRule, FlowType :: Success*)
25: **end for**
26: **return** *T*

Figure 12 introduces the whole process of the transformation validation. Figure 12a defines a success and a negative success condition that are stated against the transformation. In Figure 12b two validation points are introduced. These validation points define where and which conditions should be validated. Figure 12c1 depicts the validation rule created based on condition *ConditionSC* and Figure 12c2 shows the validation rule created based on condition *ConditionNSC*. In Figure 12d1 the original transformation is presented. Figure 12d2 shows the state after extending the transformation with a flow final and the validation rule *RuleSC*. The flow edge between *RuleSC* and *Rule3* is followed in the case of successful validation, otherwise the control follows the new edge starting from *RuleSC* and pointing to flow final node. Finally, Figure 12d3 presents the state of the control flow after extending it with the validation rule *RuleNSC*. The flow edges are connected in a different way: in the case of satisfying the conditions of *NSC* the *successful* edge is followed that points to the flow final, otherwise the control is passed to *Rule2*.

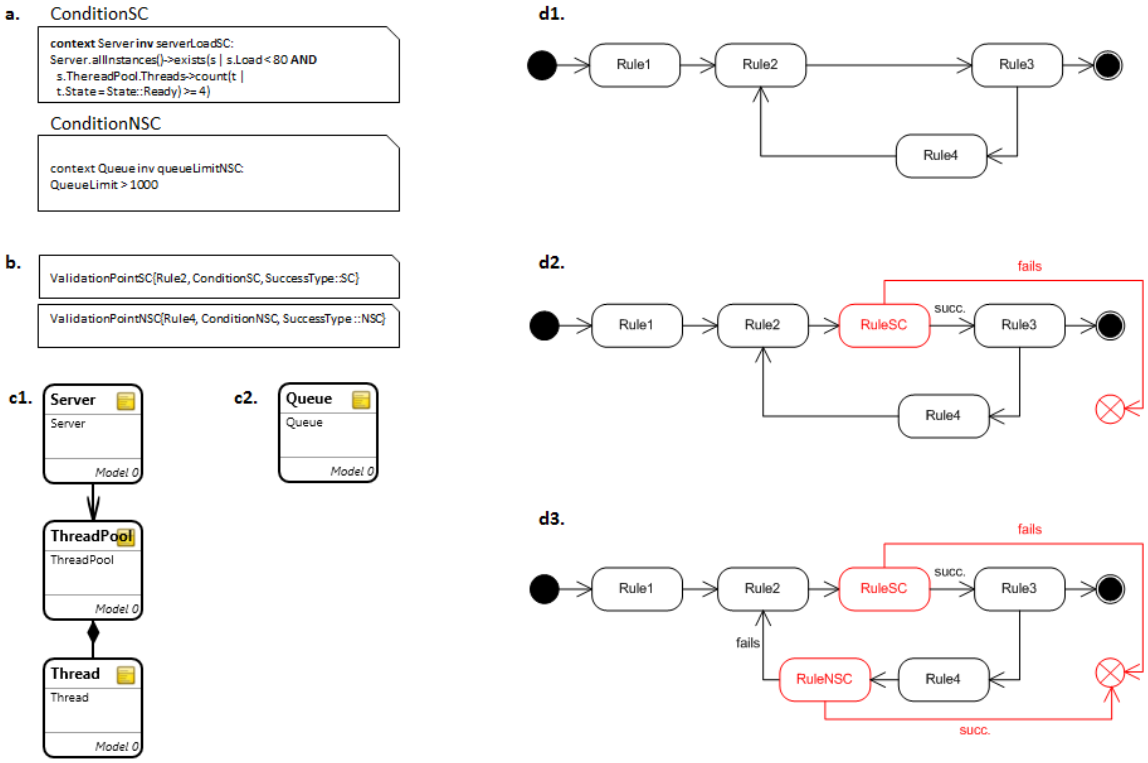


Figure 12. Algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES: (a) A success and a negative success condition of the transformation, (b) Validation points, (c) Generated validation transformation rules (*RuleSC* and *RuleNSC*), (d) The stages of the transformation controlflow extension.

The following propositions formulate statements related to the presented algorithms, the transformations processed with these algorithms and the models resulted by the extended transformations.

Proposition 3. Assume an input model *H*, a model transformation *T*, and a success condition *SC*. A validation transformation rule *VR* is generated from *SC* with the algorithm GENERATEVALIDATIONTRANSFORMATIONRULE. Model transformation *T* is extended with the algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES, using the validation point *VP{T.LastTransformationRule, SC, SuccessType::SC}*. The resulted transformation is *T'*. If the transformation *T* validates the success condition *SC* for the input model *H*, then transformation *T'* also validates the success condition *SC* for the input model *H*. Transformations *T* and *T'* generate the output models *M* and *M'* respectively.

Proof. Transformations *T* and *T'* differ only with the validation transformation rule *VR*: transformation *T* does not contain the validation rule *VR*, but transformation *T'* includes the validation rule *VR* as the last transformation rule in the control flow model. According to the assumptions, transformation *T* validates the success condition *SC* for the input model *H*. This means that, after executing the transformation *T* for the input model *H*, the conditions defined by success condition *SC* are satisfied by the output model *M*. The validation transformation rule *VR* is generated based on the success condition *SC,* thus *VR* validates the same conditions. Therefore, the output model *M'*, generated by the transformation *T'*, also satisfies the success condition *SC*.

Proposition 4. Assume an input model *H*, a model transformation *T*, and a negative success condition *NSC*. A validation transformation rule *VR* is generated from *NSC* with algorithm GENERATEVALIDATIONTRANSFORMATIONRULE. Model transformation *T* is extended with algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES using the validation point *VP{T.LastTransformationRule, NSC, SuccessType::NSC}*. The resulted transformation is *T'*. If the transformation *T* validates the negative success condition *NSC* for the input model *H*, then transformation *T'* also validates the negative success condition *NSC* for the input model *H*. Transformations *T* and *T'* generate the output models *M* and *M'* respectively.

Proof. Based on the proof of *Proposition 3*.

Proposition 5. Assume an input model *H*, a model transformation *T*, a success condition *SC* and a validation point *VP{T.LastTransformationRule, SC, SuccessType::SC}*. Based on the success condition *SC*, a validation rule *VR* is generated with algorithm GENERATEVALIDATIONTRANSFORMATIONRULE. The transformation *T* is extended by algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES, applying the validation point *VP*. The resulted transformation is *T'*. If the transformation *T'* finishes successfully for input model *H,* generating the output model *M'*, then the original transformation *T* validates the success condition *SC* for input model *H* while generating the output model *M*.

Proof. Based on the assumptions, transformation *T'* is executed successfully for the input model *H*. This means that after executing the transformation *T'* for the input model *H*, the conditions defined by success condition *SC* are satisfied on the output model *M'*. This is validated by the validation rule *VR*. Transformations *T* and *T'* differ only with the validation transformation rule *VR*. A validation rule does not make any modifications; its only role is to validate the required conditions (*Proposition 1*). Therefore, for input model *H,* the execution of transformation *T* is successful and the resulted output model *M* satisfies the success condition *SC*. This means that transformation *T* validates the success condition *SC* for input model *H*.

Proposition 6. Assume an input model *H*, a model transformation *T*, a negative success condition *NSC*, and a validation point *VP{T.LastTransformationRule, NSC, SuccessType::NSC}*. Based on the negative success condition *NSC*, a validation rule *VR* is generated with algorithm GENERATEVALIDATIONTRANSFORMATIONRULE. The transformation *T* is extended by algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES applying the validation point *VP*. The resulted transformation is *T'*. If transformation *T'* finishes successfully for input model *H* generating the output model *M'*, then the original transformation *T* validates the negative success condition *NSC* for input model *H* while generating the output model *M*.

Proof. Based on the proof of *Proposition 5*.

Proposition 7. Assume that a transformation *T* terminates for an input model *H*. Applying the algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES we extend the transformation *T* with one or more validation transformation rules. The resulted transformation is *T'*. Transformation *T'* terminates for input model *H*.

Proof. The proposition states that extending a model transformation *T* using the algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES does not make the resulted transformation *T'* non-terminating. The generation of the validation rules is performed by algorithm GENERATEVALIDATIONTRANSFORMATIONRULE. Based on *Proposition 1*, we know that a validation transformation rule does not modify the output model. The extension of the transformation *T* is performed by algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES. The algorithm does not insert loops into the control flow model of the transformation. All of the validation rules

added to the transformation are executed once. Irrespectively of their execution result, the execution of the inserted validation rules requires a final number of steps.

If a validation transformation rule is performed successfully, then the control of the transformation is not affected. Otherwise, if the validation transformation rule was unsuccessful, then the transformation terminates. Therefore, because of the validation errors, a validation transformation rule can cause an originally non-terminating transformation to terminate.

We can see that these types of rules (do not perform modification) and this type of transformation extension (no additional loops and the inserted rules are performed only once) do not make the originally terminating transformation *T* non-terminating.

Proposition 8. Assume an input model *H*, a model transformation *T*, a success condition *SC*, a negative success condition *NSC*, and two validation points VP_{SC}{*T.LastTransformationRule, SC, SuccessType::SC*} and VP_{NSC}{*T.LastTransformationRule, NSC, SuccessType::NSC*}. The transformation *T* is extended by the algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES applying the validation points VP_{SC} and VP_{NSC}. The resulted transformation is *T'*. The generated validation rules for *SC* and *NSC* are R_{SC} and R_{NSC} respectively. Transformations *T* and *T'* are executed for the same input model *H*, the output models are *M* and *M'* respectively. If the rules R_{SC} and R_{NSC} successfully validate the conditions, i.e. *SC* is present in the model and *NSC* cannot be found in the model, then the resulted output models *M* and *M'* are identical.

Proof. The proposition states that, in the case of successful validation, the inserted validation rules do not have any effect on the final output model.

Validation transformation rules (R_{SC} and R_{NSC}) perform the validation of the required conditions (*SC* and *NSC*). According to the algorithm EXTENDTRANSFORMATIONWITHVALIDATIONRULES, if validation rules successfully validate the conditions, then via a *Success* type flow edge the control is passed to the next transformation rule. This means that validation rules inserted between two original transformation rules do not modify either the control flow path or the output model. Therefore, transformations *T* and *T'* perform the same operations and transformation *T'* does additional validation. The result is, in the case of successful validations, the output models *M* and *M'* are identical.

Note that in the case of validation error the output models *M* and *M'* can be different. The reason is that unsuccessful validation rules stop the transformation execution and all rules that follow an unsuccessful validation rule in the control flow model are not executed.